

A Fast Dynamic Compression Scheme for Low-Latency On-Chip Address Buses

Jiangjiang Liu
Dept. of Comp. Sci.
Lamar University
Beaumont, TX 77710, U. S. A.
E-mail: liu@cs.lamar.edu

Krishnan Sundaresan
Dept. of Elect. & Comp. Eng.
Michigan State University
East Lansing, MI 48824, U. S. A.
E-mail: sundare2@egr.msu.edu

Nihar R. Mahapatra
Dept. of Elect. & Comp. Eng.
Michigan State University
East Lansing, MI 48824, U. S. A.
E-mail: nrm@egr.msu.edu

Abstract—As implementation technology scales down, interconnects become a major impediment to improving performance and reducing cost. In this paper, we present a new compression scheme, called partial match compression, to compress addresses dynamically and also analyze how area slack arising from compression can be exploited for bus latency improvement by increasing inter-wire spacing. Our results show that wire delay and costs can be reduced significantly using this approach. Our proposed compression technique uses a small cache of size not bigger than 500 bits at the sending end and registers at the receiving end and helps reduce the number of on-chip address-carrying lines significantly. Compared to a previously proposed scheme called bus expander, performance can be improved up to 3.13% when addresses are compressed with our technique. We also investigated wire spacing to exploit the area slack created due to compression and found that, on the average, it can reduce wire delays by up to 83.17% and thereby improve program performance by about 16%.

I. INTRODUCTION

As implementation technology scales down, integration densities are increasing. But, as on-chip wiring complexity grows exponentially with gate count, chip design is becoming increasingly interconnect-centric. Trends have also shown that integrated circuit (IC) pin count is growing at 8% to 11% per year, necessitating more and more on-chip I/O pads, which makes die area pad-limited [1]. Also, wire delays are severely limiting the use of higher clock frequencies in high-performance designs. For instance, in the Pentium 4 microprocessor, designers found that inter-wire capacitive coupling in global wires limited the clock frequency by 50 MHz from the maximum possible for that technology (0.18 μm) [2]. Also, in the Pentium 4 microarchitecture, two out of the twenty pipeline stages are devoted exclusively to carrying signals on buses [3]. Thus, wires have become a major impediment to realizing the performance, energy, and cost gains that motivate technology scaling. This will be more so in the future.

Various approaches have been adopted at different levels of abstraction to keep problems due to wire delay in check. Some of them include: new materials such as Copper interconnect and low-K dielectric insulators; high-aspect ratio (tall and thin) wires at the process level; use of timing- instead of area-driven routing algorithms at the layout level; and repeater and flip-flop insertion and simultaneous wire and driver sizing at the circuit level. But, in spite of applying these techniques,

wires still pose a major problem for current designs [4]. An effective solution to alleviate these problems may be to consider microarchitecture-level techniques which provide more room for optimization. This is the focus of our work.

The organization of the rest of this paper is as follows. We describe related work and our contributions in Sec. II. Next, in Sec. III, we discuss our proposed techniques. Then, in Sec. IV, we describe our simulation environment and methodology. In Sec. V, we present results and discussions. Finally, we conclude in Sec. VI.

II. RELATED WORK

Address buses have been studied widely in previous work and schemes have been proposed to reduce their energy and/or area/cost. The potential for compression of addresses in microprocessors using entropy models has been studied in previous work [5], [6]. Dynamic compression schemes, dynamic base register caching (DBRC) and bus expander (BE), that use small compression caches at the sending end and register files at the receiving end of a processor-to-memory address bus to reduce off-chip bus widths and pin counts were first proposed in [7], [8] and subsequently used for compressing instruction and data buses in [9]. A recent attempt at using compression to reduce on-chip wire delay is presented in [10]. However, in that work, wire delay reduction was treated in an ad hoc manner and the performance impact was estimated with first-order approximation and not using simulations [10].

Most recently, a comprehensive analysis and comparison results for the dynamic address compression schemes, DBRC and BE, from the perspective of optimizing performance, energy, and cost, was presented in [11]. In that work, we reported the optimal compression cache parameters that result in minimum extra cycle penalty and the corresponding energy savings, compression cache miss rates, and address compression ratios for a wide range of compressed bus widths for both DBRC and BE. Also, we presented a number of techniques that can be used with BE for address buses to improve the energy efficiency when transmitting compressed addresses over narrower-width buses [12].

A. Contributions

In this work, we propose a new microarchitecture-level approach using which information can be compressed or decompressed in a dynamic manner and transmitted on narrower buses with substantially lower overall cost and with performance improvements. To achieve this, we propose a technique called *partial match compression* (PMC). Then, we show how the area slack created due to compression can be exploited to optimize inter-wire spacing and thus wire coupling energy. Our work is perhaps the first to propose techniques that target reductions in wire latency and cost of address buses simultaneously and examine its impact on overall program performance using a realistic simulator and real-world benchmark programs.

The PMC scheme presented in this work is somewhat tailored to address buses. We chose to apply our scheme only to address buses in this work to demonstrate its effectiveness and the utility of our methodology. In future work, we intend to develop similar schemes for instruction and data buses. Even otherwise, schemes proposed in this work are applicable to instruction and data buses. It is well-known that address information is the most predictable and mostly strided at the processor to level-one cache (P→L1) level. Hence, in this work, we apply compression to the address bus between the level-one (L1) and level-two (L2) caches, where address references are more unpredictable, though still compressible, as explained next. Our analysis has shown that the address traffic between L1 and L2 caches in a processor-memory system has substantial amounts of both *temporal* and *value redundancy*, the former, which implies the presence of idle cycles or timing slack due to unequal intervals of time between L1 cache misses, and the latter, which implies repeated information or parts thereof due to the fact that accesses to L2 cache follow certain locality characteristics [13]. Further, at the L1→L2 level, address sequentiality is reduced due to interspersed instruction and data address references to the unified L2 cache and due to hits at the L1 caches.

The important results presented in this work are highlighted next. By using a new partial-match compression technique that uses a small cache of size not bigger than 500 bits at the sending end and registers at the receiving end, we compress address streams dynamically. This helps reduce the number of on-chip address-carrying lines significantly. We show that compression cache miss rate, wire delay, and costs can also be appreciably reduced using our technique. Compared to a previously proposed scheme called bus expander [9], performance can be improved up to 3.13% when addresses are compressed with our technique. We also investigated wire spacing to exploit the area slack created due to compression, and found that, on the average, such techniques can reduce wire delays by up to 83.17% and consequently improve program performance by about 16%.

B. Benefits and Impacts

Some of the benefits of bus compression and its impacts are described next. First, using compression, the amount of on-

chip bus hardware, including number of bus lines, associated driver and receiver circuits, and repeaters, can be reduced. If the bus connects to off-chip hardware, the number of I/O pads and buffers and pins can also be reduced. These hardware reductions can result in substantial area and cost savings. Second, substantial energy/power savings can be obtained in on- and off-chip interconnects with bus compression, since, due to the smaller area, bus capacitance reduces. Further, by using area no more than a normal uncompressed bus, a narrow bus can: (1) use greater spacing between bus lines, which will reduce inter-wire capacitance and hence delay, power consumption, and crosstalk; and/or (2) use wider wires to reduce resistance and hence delay and potentially improve performance. Also, since bus latency improves with wire spacing, timing slack is created and this can be exploited (via, e.g., slack passing) to reduce power consumption by lowering voltage. Third, performance benefits can be potentially obtained, not only by wire spacing and wire-width optimization as mentioned above, but also by introducing shielding wires to reduce inter-wire crosstalk (again, without using area more than original bus) in compressed buses, since all of these reduce propagation delay of wires.

Although a compression scheme may itself entail some performance, area, and power consumption overheads due to extra logic, these overheads will not be much compared to the savings potentially obtained by compressing long on-chip and off-chip buses. This is because the size, speed, and power consumption of logic (which will be used to do compression and decompression) scale better than those of interconnect (which will be used to communicate the information) and hence these overheads will continue to decrease over time. In addition, our bus compression technique can reduce bus power consumption—which we will analyze in future work—and also the cost by reducing the bus width with very little performance penalty, whereas existing techniques like bus encoding schemes can only reduce bus energy with extra cost for the added control lines.

III. PROPOSED WORK

In the next subsection, we first describe how bus compression works and discuss the shortcomings of existing schemes in order to motivate our proposed techniques which will be discussed later in Sec. III-B. The details of previous bus compression techniques can be found in [7], [8], [11], [12].

A. Limitations of Previous Bus Compression Schemes

The structure and working of a dynamic bus compression technique, such as BE, is explained next. As shown in Fig. 1(a), the higher-order portion of an address is compressed by the compressor which is implemented as a small *compression cache*. The lower-order portion, which is not very compressible due to its highly-varying nature (U field in Fig. 1(b)), is transmitted as is on the compressed bus. The width of the compressed bus is equal to the width of the compressed address. At the receiving end, the original address is retrieved by looking up a register file present in the

decompresser hardware. The small compression cache stores the higher-order portion, called tag field (T field in Fig. 1(b)), of recently transmitted addresses in its tag-RAM.

To compress addresses, a set of bits from the incoming address, called the *index* (I field in Fig. 1(b)), is used to search the compression cache. Note that, in this paper, we always refer to tag and index fields for the compression cache, and this is not related to the tag and index fields of conventional (L1 or L2) caches in the memory system. If the compression cache is n -way set-associative, then one of n tags stored in the set indexed by the I field can potentially fully match the tag from the incoming address, and provide the *way bits* (W field in Fig. 1(b)).

In the case of a hit, the I and W fields, along with the hit/miss control bit (C field) of the tag and the U field will form a compressed address with a combined bit width less than the original. The compressed bus width is equal to the sum of of the widths of the C, I, W, and U fields. So in the hit case, there is no performance penalty due to the transmission of the compressed address. In case of a miss at the sending end, in the compression cache, the tag corresponding to the least recently used (LRU) entry in the set indexed by the I field is replaced by the tag of the new address. The compressor transmits the C field and the entire address (starting from the higher-order to lower-order bits), which is wider than the compressed bus width. The decompressor will be updated at the receiver end accordingly. The miss in the compression cache causes miss penalty for transmission because extra cycles will be needed for transferring the control field and also the entire address.

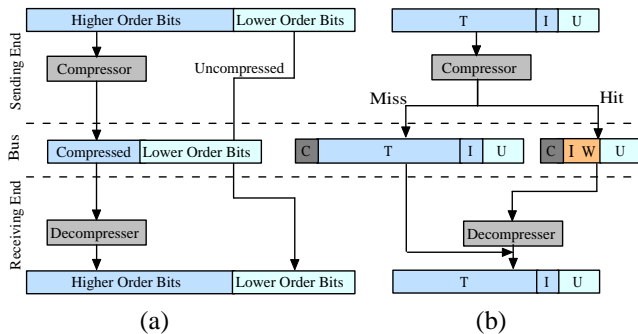


Fig. 1. Bus Compression Scheme: (a) General organization of a bus compression scheme. (b) Figure depicting how compressed addresses are formed.

In earlier work, we reported the optimal compression cache parameters and proposed a number of transmission techniques to improve the original BE address compression scheme by considering trade-offs between performance, on-chip and off-chip energy consumption, and cost [11], [12]. We shall refer to this as *optimized BE* and will use it as the basis for comparison. The L1→L2 address bus that we consider for compression has an uncompressed width of 38 bits. Results for optimized BE are summarized in Table I. We report results for three categories of bus widths: *narrow* (compressed buses with 8, 10, or 12 lines), *medium* (compressed buses with 14 to 16 lines), and *wide* (compressed buses with 19 to 32 lines).

B. Partial Match Address Compression

In this subsection, we describe the partial match compression (PMC) cache. As the name suggests, in this cache we allow partial matches to register as compression hits. Thus, we check for the longest match between the tag portion stored in the compression cache and the tag portion of the incoming address. Since there is more redundancy in the higher-order bits of the address, we consider k possible groups of bits (shown in Fig. 2 as part 0, part 1, and part 2 for $k = 3$) ending at the most significant bit (MSB) of the incoming address as candidates for partial match. There are three main hit/miss cases in PM as explained next.

- Complete hit: If the tag matches fully, PM performs exactly the same as the BE scheme studied in prior work [9], [11], [12]. The control bit C_H is set high to indicate a complete hit. If not, it is set low.
- Partial hit: If a partial match in any of the k groups occurs, the control pattern corresponding to the longest match is transmitted to indicate a partial hit. For example, when $k = 3$, control pattern C_1C_0 will be 00, 01, or 10 for part 0 hit, part 1 hit, or part 2 hit, respectively. The remaining portion of the higher-order part of the address (that did not match the tag), T_{miss} , is sent in uncompressed form as is the lower-order portion of the address. In this case, less bits are transmitted over the compressed bus than would have been in optimized BE; this reduces the power consumption for transmission.
- Complete miss: In case of a complete miss in PM compression cache, when none of the partial matches succeed, the entire address is sent along with the control bits $C_H = 0$ and $C_1C_0 = 11$. Even though more bits need to be transferred in PM than in the case of optimized BE—due to the extra control bits for indicating a complete miss—the performance will not be affected much because the number of complete misses will be much less in PM than in the case of optimized BE. We discuss this further in Sec. V-A.

It is to be noted that our proposed PMC cache’s tag field is updated whenever complete matches do not occur, which is the same as in the BE compression cache. So the tags stored in PM and optimized BE compression caches are identical when running the same program. Hence, the compression cache power consumption and latencies are also likely to be identical for similar-sized units.

When compression/decompression schemes are used in buses, a major concern is the impact on latency. We explain next why compression/decompression latencies will be minimal and may even be lower than bus encoding/decoding latencies in actual design. We use the simplest low power encoding scheme, bus-invert (BI) [14], as a representative scheme to justify this. In BI, encoding consists of three steps. First, the Hamming distance is computed. To do this, the current n -bit pattern and the previous n -bit pattern that was transmitted on the bus in the last cycle are bitwise exclusive-ORed (XOR) and the number of “1”s in the result is counted.

TABLE I
 COMPRESSION CACHES AND COMPRESSED BUSES FOR ADDRESS COMPRESSION USING BE SCHEME. ALL COMPRESSION CACHES ARE 2-WAY
 SET-ASSOCIATIVE.

	BusWidth (Original Address Width: 38 bits)								
	Narrow			Medium		Wide			
	8	10	12	14	16	20	24	28	32
Index (I field)	1	2	3	2	2	1	1	1	1
Tag (T field)	32	30	28	26	24	20	16	12	8
Miss Rate	0.39	0.28	0.21	0.15	0.10	0.00	0.00	0.00	0.00
Miss Penalty (cycles)	4	3	3	2	2	1	1	1	1
Perf. Penalty(%)	4.98	2.34	1.62	0.83	0.46	0.07	0.00	0.0	0.0
Comp. Cache Size (bit)	156	256	480	224	208	88	72	56	40

This step requires a constant time operation for bitwise XOR and $O(n)$ to $O(\log_2 n)$ time for counting depending upon the counter structure used. In the second step, the Hamming distance is compared with $\frac{n}{2}$ to check which is greater; this can be completed in $O(n)$ to $O(\log_2 n)$ time, again depending on the hardware structure used. Finally, the current pattern is inverted or sent as-is and this takes constant time. Thus, BI encoding takes at least $O(\log_2 n)$ time. In contrast, our optimized address compression scheme, which uses a 480-bit compression cache, has an access latency of well under one cycle according to CACTI estimates [15]. Hence, compression schemes are more performance-efficient compared to encoding.

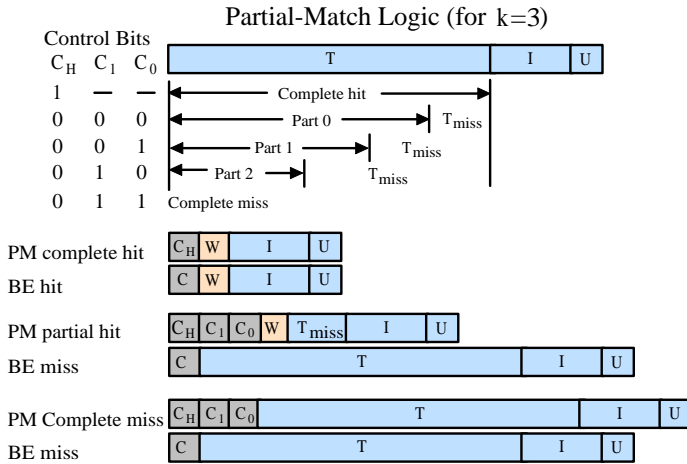


Fig. 2. Partial Match Address Compression.

IV. METHODOLOGY

We used *sim-alpha*, the validated Alpha 21264 simulator based on the SimpleScalar tool, as the platform for our experiments [16]. We simulated address compression and transmission and collected performance and bus energy results for 50 million committed instructions for 7 integer benchmarks and 7 floating-point benchmarks randomly chosen from the SPEC CPU2000 suite: *gcc*, *gzip*, *parser*, *vpr*, *twolf*, *mcf*, *crafty*, *applu*, *swim*, *wupwise*, *lucas*, *art*, *ammp*, and *equake*. We implemented our bus compression scheme for the 38-bit

address bus between the L1 and L2 caches, in a memory hierarchy with two levels of caches and a main memory. In our simulated system, the L1→L2 address bus carries physical addresses and both instruction and data addresses are carried on the same bus. In order to compare our design with optimized BE, we used the optimal compression cache parameters from our earlier work [11], summarized in Table I, for different compressed buses ranging from 8 to 32 bits.

The execution time of a benchmark program running on the target system was collected in terms of processor cycles. Since we used an execution-driven simulator, our calculation of performance included any latencies due to pipeline stalls too and not just the latencies due to address transmission. We report the performance improvement (PI) for n benchmarks which is defined as follows:

$$PI = \frac{\sum_{i=1}^n (t_{i,BE} - t_{i,PM})}{\sum_{i=1}^n t_{i,BE}}, \quad (1)$$

where $t_{i,PM}$ is the total time (processor cycles) for execution of program i on the modified target system with PM and $t_{i,BE}$ is the total time for execution of the same program on the target system with BE.

A. Delay Model

Next, we describe the wire delay model used in this work. Delay characteristics of buses depend on self and coupling transitions occurring in individual wires and pairs of adjacent wires, respectively, constituting the bus. Self transitions are of two types: charge ($0 \rightarrow 1$) and discharge ($1 \rightarrow 0$), and coupling transitions in a pair of adjacent wires are of three types: *coupling charge transitions* ($00 \rightarrow 01^1$, $00 \rightarrow 10$, $01 \rightarrow 11$, and $10 \rightarrow 11$), *coupling discharge transitions* ($01 \rightarrow 00$, $10 \rightarrow 00$, $11 \rightarrow 01$, and $11 \rightarrow 10$), and *toggle transitions* ($01 \rightarrow 10$ and $10 \rightarrow 01$).

Due to inter-wire capacitive coupling, the propagation delay of the k^{th} -wire in a bus, which is a function of transitions in its neighboring wires $k-1$ and $k+1$, can be expressed as follows [1]: $t_{p,k} = g \cdot C_w \cdot (0.38R_w + 0.69R_D)$, where C_w is the self capacitance (capacitance to ground) and R_w and R_D are the wire and driver resistances of the wire, respectively; g is

¹For two lines i and j , this notation represents the transition: $V_i^{initial} V_j^{initial} \rightarrow V_i^{final} V_j^{final}$.

the *delay correction factor* due to crosstalk between adjacent wires separated by the minimum spacing distance and is a function of the *capacitance ratio* $r = C_c/C_w$ (C_c is the coupling capacitance between two adjacent wires).

TABLE II
WIRE DELAY CORRECTION FACTORS.

$k-1, k, k+1$	Delay factor (g)
$\uparrow, \uparrow, \uparrow$	1
$\uparrow, \uparrow, -$	$1+r$
$\uparrow, \uparrow, \downarrow$	$1+2r$
$-, \uparrow, -$	$1+2r$
$-, \uparrow, \downarrow$	$1+3r$
$\downarrow, \uparrow, \downarrow$	$1+4r$

Note that the value of r depends on technology and the layer of metal being considered. For our simulations, we used $r \approx 5$ which was obtained for topmost metal layer (M6) in TSMC 0.18 μm technology considering the substrate layer as the ground plane. The effect of non-standard wire spacings can be captured in the above model by introducing a *spacing correction factor* for r in the above equations and Table II. Thus, if v_0 is the minimum allowable spacing between two wires and v is the new spacing ($v \geq v_0$), then the new capacitance ratio will be: $r \cdot (\frac{v_0}{v})$. In this case, the worst-case delay of a wire (which occurs when there are toggles on both adjacent coupling capacitances of the wire–bottommost row in Table II) will improve by a factor of $\frac{1+4r}{1+4r \cdot (\frac{v_0}{v})}$.

V. RESULTS AND DISCUSSION

In this section we present results showing how PM compression performs compared to bus expander proposed in prior work.

A. Feasibility of Partial Match

To analyze how PMC can help reduce compression cache miss rate and to explore an effective way to partition the tag field, we collected the individual hit frequency (IHF) for each possible partition ending at the MSB for all 14 benchmark programs listed earlier. For each partition, we collected the number of complete hits and complete misses out of all references to the PM compression cache during the simulation as shown in Fig. 3 for an n -bit tag field for different compressed buses. For an n -bit tag field, the possible partitions are n bits, $n-1$ bits, \dots , 1 bit, and 0 bit ending at MSB. If the entire tag field in the current address hits in the PM compression cache, it is called *complete hit* or *full match*. If the matching length is smaller than n bits but larger than 0 bits, it is called *partial hit* or *partial match*. In the partial hit case, out of all entries in a set pointed to by the address index, the entry which has the longest matching length is the hit entry. If the matching length is 0, it is a *complete miss*.

Fig. 4 shows the IHF of various partitions for different compressed bus widths (4-bit, 6-bit, \dots , 30-bit, 32-bit). As the compressed bus width decreases, the frequency of complete hit decreases, which means PMC will be more effective for

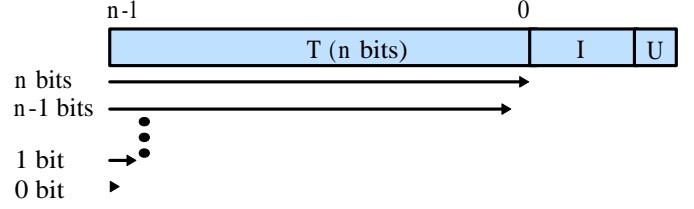


Fig. 3. All Possible Partitions Ending at MSB.

narrower compressed buses. In a scheme like BE, there are only two outcomes for a compression cache lookup: complete hit and complete miss. So even if the tag field from the incoming address and the stored tag differ in a single bit, a miss will occur and the entire tag field will be transmitted over the compressed bus in multiple cycles. On the other hand, in PMC, only the unmatched bits need to be transferred if a partial hit occurs during the lookup, which will reduce the number of extra cycles needed. For example, for an 8-bit compressed bus, the higher-order 32 bits are used as tag for compression, the miss rate is 0.39 in BE (Fig. 5, column 3). For this case, 9% of the misses were found to be due to a one bit mismatch (column labeled [1, 1] in Fig. 4), 15% due to either a one- or two-bit mismatch (column labeled [1, 2]), 30% due to mismatches in less than or equal to 5 bits (column labeled [1, 5]), and 0% due to a complete miss in all bits of the 32-bit tag. As we observe, a complete miss will be a very unlikely event when partial hits are considered. Therefore, if PMC is applied to the higher-order 27 bits in the tag, 30% of the misses now need to transfer only a maximum of 5 bits and the control bits, which is much less than the original width of the tag. A percentage summary of partial match hit at different partitions for narrow and medium compressed buses is shown in Fig. 5.

In addition, we consider the miss penalty when choosing a combination of different partitions. In the case of complete hit, only one cycle is needed for the transmission, which is the same as BE. Miss penalty (MP) is the extra cycles taken due to partial hit or complete miss. The performance of address compression also depends on the miss penalty of different partitions. Based on our analysis of IHC and MP for different partitions, we selected the number of partitions and the partition points for different compressed buses. Table III lists the partition points and corresponding miss penalties used in our simulations. For example, three partitions are used for an 8-bit compressed bus. The tag length is 32 bits. Ending at the MSB, part 0 has 22 bits with $MP = 4$, part 1 has 14 bits with $MP = 3$, and part 2 has 6 bits with $MP = 2$.

B. Performance Comparison of PM and BE

We report the miss penalties incurred during BE and PM address compression to facilitate a side-by-side comparison. Since there are several partitions in PM, as listed in Table III, the PMC miss penalty is calculated as the weighted average of the miss penalties of all partitions (the weights depending

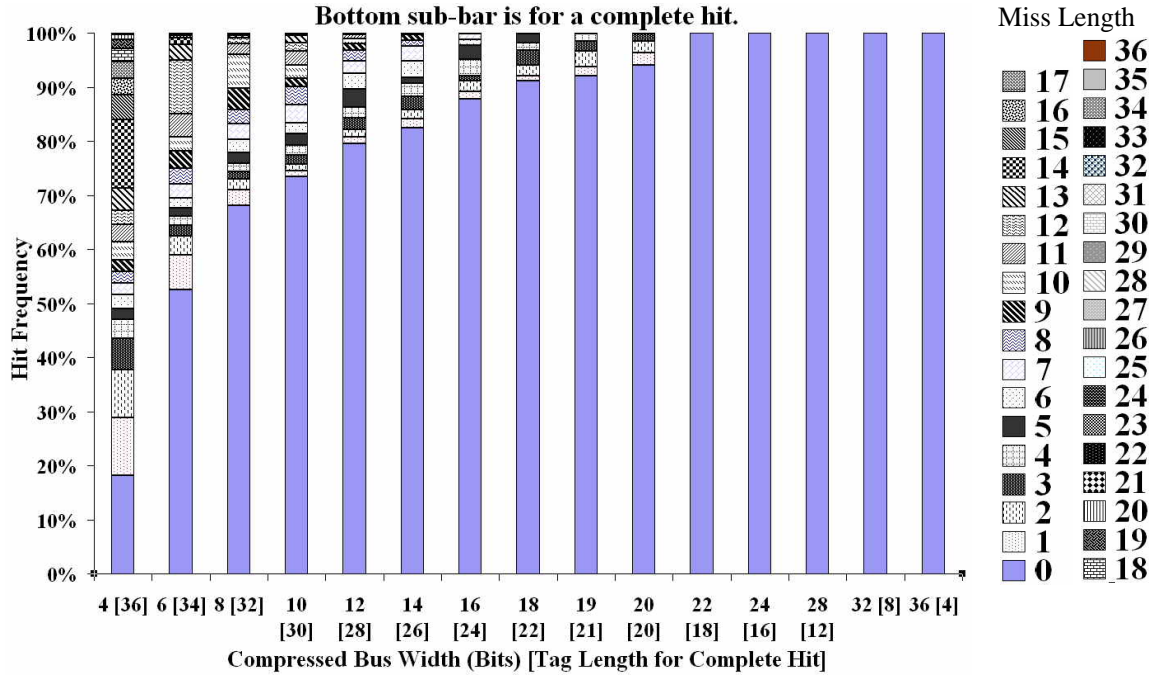


Fig. 4. Frequency of Full and Partial Tag Matches.

Bus Width (Bits)	Tag (Bits)	BE Miss Rate	Percentage of Partial Hits In Tag Out of Misses in BE (%)													
			([X,Y] means min. X bits in tag missed and max. Y bits in tag missed.)													
			[1,1]	[1,2]	[1,3]	[1,4]	[1,5]	[1,6]	[1,7]	[1,8]	[1,9]	[1,10]	[1,11]	[1,12]	[1,13]	[1,14]
4	36	0.82	13	24	31	35	38	41	43	46	49	53	57	60	65	80
6	34	0.47	13	21	25	29	32	36	41	47	54	60	69	89	96	98
8	32	0.39	9	15	19	24	30	38	47	55	68	88	94	97	99	99
10	30	0.28	4	9	15	22	30	37	50	63	68	78	88	93	98	99
12	28	0.21	6	13	23	33	49	63	75	85	90	95	98	99	99	99
14	26	0.15	9	19	33	47	53	70	86	92	98	99	99	99	99	99

Fig. 5. Converting Complete Misses in BE to Partial Hits in PM.

TABLE III

PARTITION POINTS FOR PM: NUMBER SHOWN IN PARENTHESES IS THE MISS PENALTY INCURRED.

Bus width (bits)									
8	10	12	14	16	18	19	20	24	32
6 (2)	9 (2)	11 (2)	13 (2)	15 (2)	17 (2)	18 (2)	20 (2)	16 (2)	8 (2)
14 (3)	18 (3)	28 (4)	26 (3)	24 (3)	22 (3)	21 (3)	-	-	-
22 (4)	30 (5)	-	-	-	-	-	-	-	-
32 (6)	-	-	-	-	-	-	-	-	-

upon the frequencies of different partial hits) for a given bus width. As shown in Fig. 6(a), compared to BE, the miss penalty can be reduced by 50%-66% with PM, especially for narrower compressed buses. Next, in Fig. 6(b), we show the average percentage of extra cycles taken to complete program execution for BE and PM compared to that for the original default system without compression for different bus widths. Again, as we see, the performance overhead due to compression reduces significantly (0.13% – 3.13%) by using PM instead of BE, especially at narrower bus widths. Since,

there was not much improvement for wider compressed buses, these results are not shown. This is expected since PM is designed for improving compression cache performance for narrower compressed buses for which BE’s miss rates are significant.

C. Bus Latency vs. Bus Area

Using compressed buses grants an extra degree of freedom while performing global wire routing for high-performance designs. Common optimizations like *net shielding* (inserting

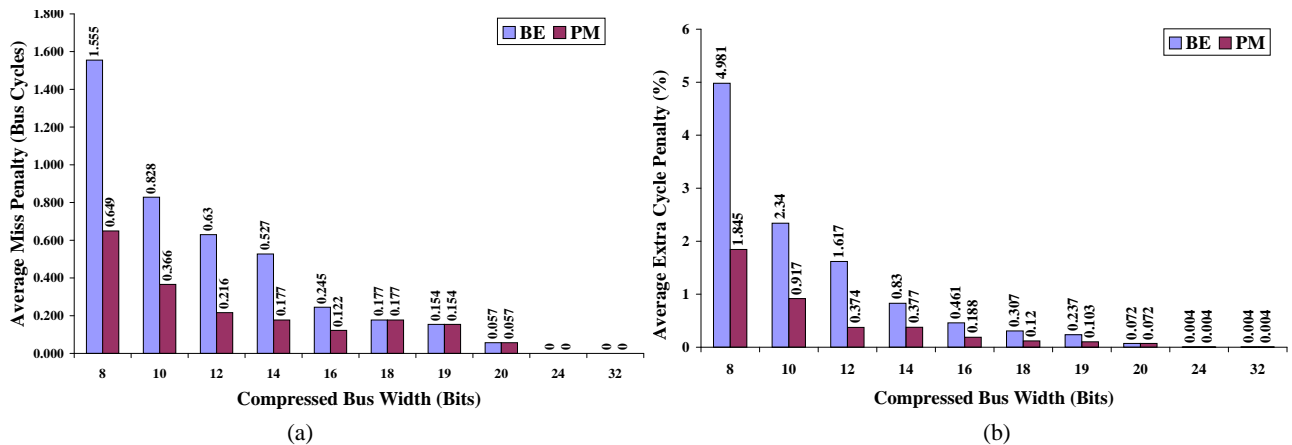


Fig. 6. Performance Comparison of PM and BE Address Compression Across Different Bus Widths: (a) Average miss penalty. (b) Average extra cycle penalty.

power or ground wires on both sides to protect a wire on the critical path from inter-wire couplings) and *soft spacing* (a technique that automatically maximizes spacing between tightly packed wires within given area constraints) can be greatly facilitated by using compressed buses. Such optimizations go a long way in achieving signal integrity and timing closure in current nanometer designs [4]. Although these techniques have been used in the VLSI design community for a long time, our work is the first to examine their implications in the context of compressed buses. In the simplest application of wire spacing in bus compression, the individual wires can be spaced further apart while maintaining the same area footprint as the original bus. Due to the increased inter-wire distance, coupling capacitances will have lower values. Also, wire spacing involves no additional cost.

The reduction in peak crosstalk delay (bus latency) with wire spacing applied is plotted for different compressed bus widths in Fig. 7. The bus delay model described in Sec. IV-A was used. In Fig. 7, the vertical bars denote wire delay ratios for the compressed bus using the proposed PM compression scheme and the x-axis labels (20%, 30%, ..., 100%) denote how much of the area footprint of the original bus is used for the compressed bus as a whole (100% means that the original and the compressed buses occupy the same total routing area). Since the delay ratio for the compressed bus with minimum spacing is unity, it is not reported in this figure. The results for all compressed buses we considered, 8-bit, 10-bit, 12-bit, 14-bit, 16-bit, 19-bit, 20-bit, 24-bit, and 32-bit, show that bus delay can be reduced, on the average, by about 83.17% for narrow (8 to 12 bits) buses, 76.89% for medium (14 to 16 bits) buses, and by about 69.63% for wide (19 to 32 bits) compressed buses using PM with wire spacing. A maximum of 93% bus delay reduction can be obtained for an 8-bit bus. Fig. 8 shows the performance improvement that is obtained when PM with wire spacing (corresponding to “100%” in Fig. 7) is used. If the entire area footprint of the original bus is used to increased wire spacing, on the average, performance can be improved by up to 16% compared to the default system

without any compression. Here, the L1→L2 address bus is assumed to have a latency of 5 CPU cycles in the default system without compression and 1 CPU cycle for compressed bus widths 8-20 (since delay ratios for these bus widths in Fig. 7 are < 0.2) and 2 CPU cycles for compressed bus widths 24-32 (since their delay ratios are between 0.2 and 0.4 in Fig. 7) when PM is used.

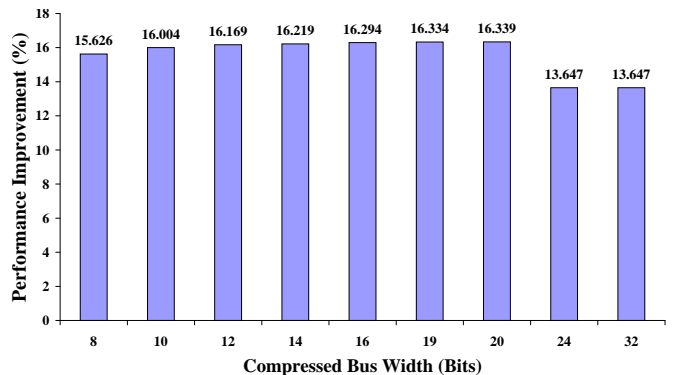


Fig. 8. Performance Improvement with Wire Spacing Across Compressed Buses for Partial Match. (The L1→L2 address bus is assumed to have a latency of 5 CPU cycles in the default system without compression and 1 and 2 CPU cycles for compressed bus widths 8-20 and 24-32, respectively, in the target system with PM.)

VI. CONCLUSION AND FUTURE WORK

In this work, we presented techniques to help reduce the growing impact of interconnects on wire delay and cost of systems. By using a new partial-match compression (PMC) technique that uses a small cache of size not bigger than 500 bits at the sending end and registers at the receiving end, we compress addresses dynamically. This helps reduce the number of on-chip address-carrying lines significantly. We showed that compression cache miss rate, wire delay, and costs can also be reduced significantly using our technique. Compared to a previously proposed scheme called bus expander, average program performance can be improved up to 3.13% when

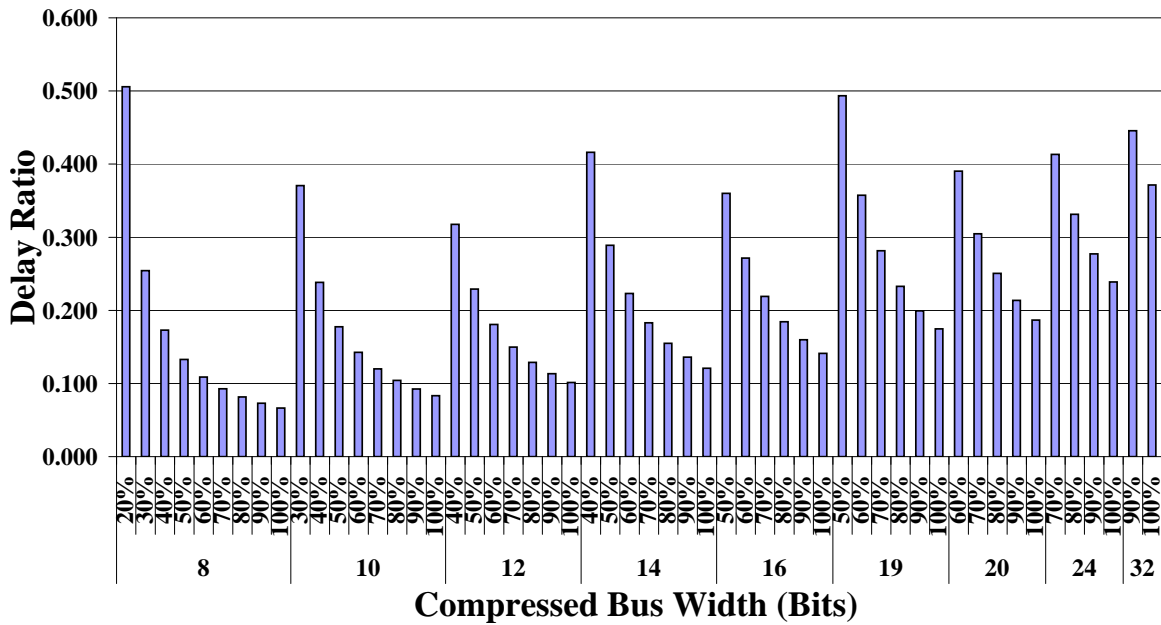


Fig. 7. Delay Variation with Wire Spacing Across Compressed Bus Widths for Partial Match.

addresses are compressed with our technique. We also investigated wire spacing to exploit the area slack created due to compression, and found that, on the average, it can reduce wire delays by up to 83.17% and consequently improve program performance by about 16%.

The PMC scheme proposed in this work can be adjusted and applied to instruction and data buses too. However, even better results can be obtained if PMC can be tailored to the characteristics of the information streams that it operates on. For example, our analysis of the potential for compression in instruction and data streams for SPEC CPU2000 benchmarks shows that compressibility varies across different fields of the instruction/data word due to the various instruction formats and data types used in a program [13]. This provides opportunities for designing a PMC scheme that matches individual fields using a separate cache for each. In future work, we intend to develop such dynamic compression schemes for instruction and data buses.

ACKNOWLEDGMENT

This research was supported by US National Science Foundation grant # 0102830.

REFERENCES

- [1] J. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits, Second edition*. Prentice-Hall, 2003.
- [2] P. Green, "A GHz IA-32 Architecture Microprocessor Implemented on 0.18 μm Technology with Aluminum Interconnect," in *Digest of Technical Papers, IEEE International Solid-State Circuits Conference*, Feb. 2000, pp. 98–99.
- [3] R. Kumar, "Interconnect and Noise Immunity Design for the Pentium 4 Processor," *Intel Technology Journal, 1st Quarter 2001*, 2001.
- [4] L. Lev and P. Chao, "Down to the Wire: Requirements for Nanometer Design Implementation," White Paper, Cadence Design Systems Inc., 2002.

- [5] D. Hammerstrom and E. Davidson, "Information Content of CPU Memory Referencing Behavior," in *Proceedings of the 4th Annual Symposium on Computer Architecture*. ACM Press, 1977, pp. 184–192.
- [6] J. Becker, A. Park, and M. Farrens, "An Analysis of the Information Content of Address Reference Streams," in *Proceedings of the International Conference on Microarchitecture*, Nov. 1991, pp. 19–24.
- [7] A. Park and M. Farrens, "Address Compression through Base Register Caching," in *Proceedings of the Annual ACM/IEEE International Symposium on Microarchitecture*, Nov. 1990, pp. 193–199.
- [8] M. Farrens and A. Park, "Dynamic Base Register Caching: A Technique for Reducing Address Bus Width," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, May 1991.
- [9] D. Citron and L. Rudolph, "Creating a Wider Bus Using Caching Techniques," in *Proceedings of International Symposium on High Performance Computer Architecture*, Jan. 1995, pp. 90–99.
- [10] D. Citron, "Exploiting Low Entropy to Reduce Wire Delay," *Computer Architecture Letters*, vol. 3, Jan. 2004.
- [11] J. Liu, K. Sundaresan, and N. R. Mahapatra, "Dynamic Address Compression Schemes: A Performance, Energy, and Cost Study," in *Proceedings of International Conference on Computer Design*, Oct. 2004, pp. 458–463.
- [12] —, "Energy-Efficient Compressed Address Transmission," in *Proceedings of the 18th International Conference on VLSI Design (VLSID), Kolkata, India*, Jan. 2005, pp. 592–597.
- [13] N. R. Mahapatra, J. Liu, K. Sundaresan, S. Dangeti, and B. Venkatrao, "A Limit Study on the Potential of Compression to Improve Memory System Performance, Power Consumption, and Cost," *Journal of Instruction-Level Parallelism*, vol. 7, July 2005.
- [14] M. Stan and W. Burleson, "Bus-Invert Coding for Low-power I/O," *IEEE Transactions on VLSI Systems*, vol. 3, pp. 49–58, Mar. 1995.
- [15] P. Shivakumar and N. Jouppi, "CACTI 3.0: An Integrated Cache Cycle Timing, Power, and Area Model," Compaq Western Research Laboratory, Tech. Rep. WRL Research Report 2001/2, Aug. 2001.
- [16] R. Desikan, D. Burger, S. Keckler, and T. Austin, "Sim-alpha: A Validated, Execution-Driven Alpha 21264 Simulator," The University of Texas at Austin, Department of Computer Sciences, Tech. Rep. TR-01-23, 2001.